EXERCISE 6
Geospatial Scripting in R

Introduction

This exercise focuses on using R in RStudio. This exercise is very fast paced, and uses much of the information that you learned in exercises 1-3 of this course. It is designed to allow you to apply many of the concepts you learned in the earlier exercises and see how to use them in a geospatial scripting environment. It also provides you with much of the information that is necessary to take part in the training, *Introduction to R and Enhancements to Geospatial Analyses* also offered by GTAC.

Objectives

- Apply and expand on what you learned about in exercises 1-3
- Learn more about R and RStudio

Required Software

- **RStudio**

Required Data

- **Course data**

Prerequisites

- *Introduction to Geospatial Scripting Exercise 1: Planning a script.*
- *Introduction to Geospatial Scripting Exercise 2: Concepts of Scripting, Part 1*
- *Introduction to Geospatial Scripting Exercise 3: Concepts of Scripting, Part 2*.

USDA Non-Discrimination Statement

In accordance with Federal civil rights law and U.S. Department of Agriculture (USDA) civil rights regulations and policies, the USDA, its Agencies, offices, and employees, and institutions participating in or administering USDA programs are prohibited from discriminating based on race, color, national origin, religion, sex, gender identity (including gender expression), sexual orientation, disability, age, marital status, family/parental status, income derived from a public assistance program, political beliefs, or reprisal or retaliation for prior civil rights activity, in any program or activity conducted or funded by USDA (not all bases apply to all programs). Remedies and complaint filing deadlines vary by program or incident.

Persons with disabilities who require alternative means of communication for program information (e.g., Braille, large print, audiotape, American Sign Language, etc.) should contact the responsible Agency or USDA's TARGET Center at (202) 720-2600 (voice and TTY) or contact USDA through the Federal Relay Service at (800) 877-8339. Additionally, program information may be made available in languages other than English.

To file a program discrimination complaint, complete the USDA Program Discrimination Complaint Form, AD-3027, found online at How to File a Program Discrimination Complaint and at any USDA office or write a letter addressed to USDA and provide in the letter all of the information requested in the form. To request a copy of the complaint form, call (866) 632-9992. Submit your completed form or letter to USDA by: (1) mail: U.S. Department of Agriculture, Office of the Assistant Secretary for Civil Rights, 1400 Independence Avenue, SW, Washington, D.C. 20250-9410; (2) fax: (202) 690-7442; or (3) email: program.intake@usda.gov.

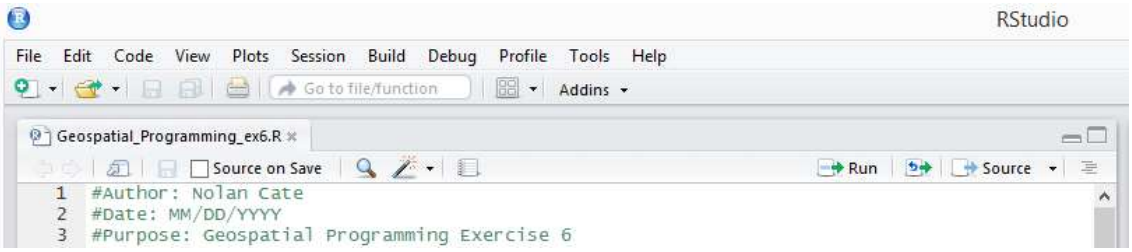
USDA is an equal opportunity provider, employer, and lender.

Table of Contents

Part 1: Preparing your Script

A. Set Up Your Script

1. Open **RStudio**
2. Save a script the same way you learned in exercise 1. Name the script **Geospatial_Scripting_ex6**. Remember to add the .R extension.
3. Now that you've saved a script in RStudio, you can edit the script in the RStudio window, in the section above the console. The first three lines of your script are going to be a header, like you learned about in exercise 1.
   i. On line 1, write the comment:
   ```
   #Author: Your Name
   ```
   ii. On line 2, write the comment:
   ```
   #Date: MM/DD/YY
   ```
   iii. On line 3, write the comment:
   ```
   #Geospatial Scripting, exercise 6.
   ```
   Your RStudio window should look something like this:

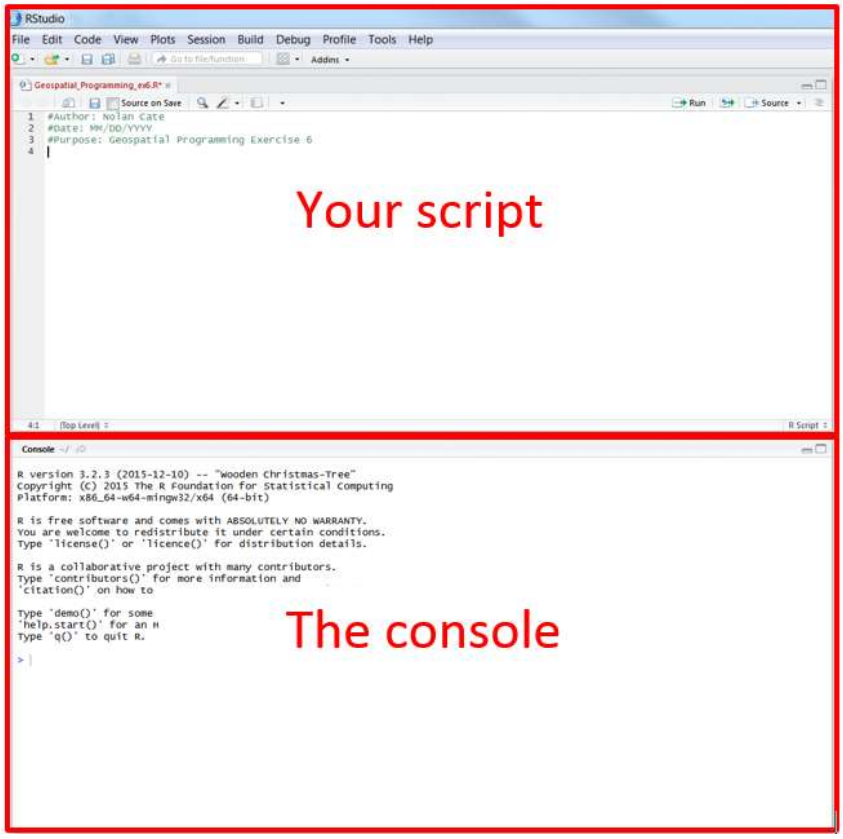


Creating headers like this will be a huge help to you or a partner at some point in the future. Including a header is always best practice when writing a script, and including comments like this throughout your script will help when running scripts later on.

B. Install Packages

Packages in R are important to understand. One of the huge benefits of a language like R is the user community, which regularly designs and releases R tools that are available to use. When an R user creates useful tools, the tools can be wrapped up in a package and accessed through RStudio. These packages are widely used, trusted sets of tools.

Because the packages are not part of the "standard" R language, they're not immediately available. We have to install the packages first. In this section of the exercise you're going to install several packages by writing several lines of code into the console, not into the script you started in the first section.



1. In the console install the package raster by typing in the line,

   `install.packages("raster")`, then hit **Enter**.
2. When the raster package has finished installing, install the rgdal package. In the console type in the line,

   `install.packages("rgdal")`, then hit **Enter**.
3. Once these packages are installed, they can be called in your script. This will let R use the tools that are contained in these packages. Back in your script, below the header that you wrote in the first section, write the line:

   `library(raster)`

   This makes it so that if you run your script again, R will load the package but you won't need to install it again.

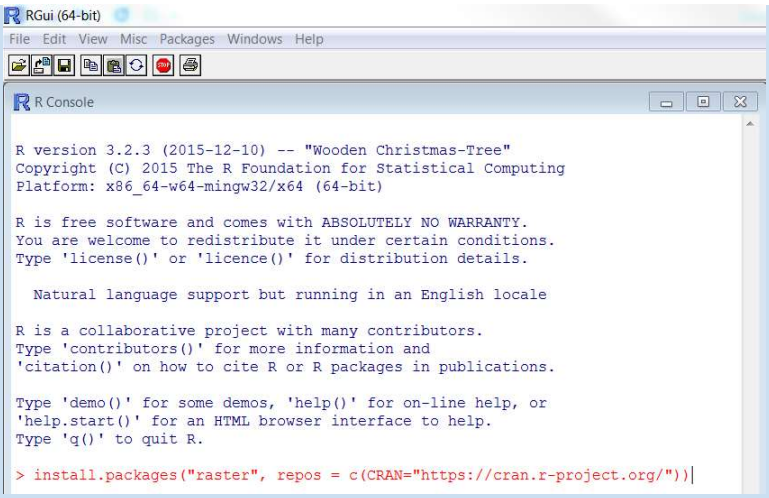
Note: *It is possible in new versions of RStudio to run into problems installing the raster package.* ***If this happens****, follow these steps:*

1. Close RStudio

2. Open a fresh R instance (not RStudio). This can be found by clicking start, then scroll down to the folder, R.

3. Run the following line of code:

install.packages("raster", repos = c(CRAN="https://cran.r-project.org/"))

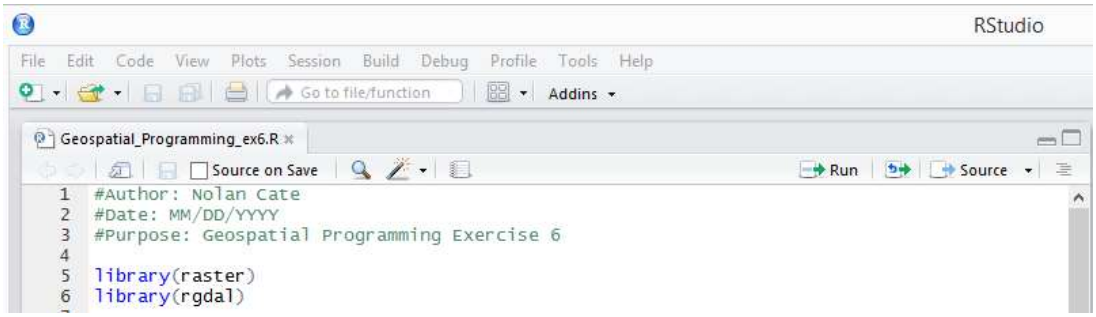


4. Close R

4. On the next line of your script, write the line:

   ```
   library(rgdal)
   ```

   Your script should now look similar to this:



5. Now you have some code that you can run. Notice the "run" button in the top right corner of your script window. If you click it, it will only run the line that your cursor is on. To run multiple lines of code, you can highlight the lines in your script you want to run, then click run. Right now, highlight all of your code, and click **Run**. You will see the comments you wrote appear in your console at the bottom of the page, plus the lines loading the packages.

Part 2: Raster Data in R

A. Exploring Your Data

In the data folder you downloaded is a Landsat tile. This single tile is made of multiple bands, all stored as a tif image. Store the data on your C drive in C:/GeospatialScripting/R (or a similar location that you will remember). The tools you'll learn about here help you explore the data in R.

1. Begin by creating the variable "path" and set it equal to C:/GeospatialScripting/R, by coding the line,

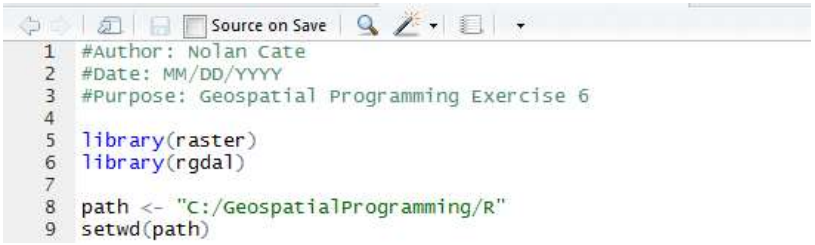
   ```
   path <- "C:/GeospatialScripting/R"
   ```

   When you run the script, this line will create a variable called path and set it equal to the string in quotes.

2. To tell R where your data is located, you are going to change your working directory. To do so, write the line,

   `setwd(path)`

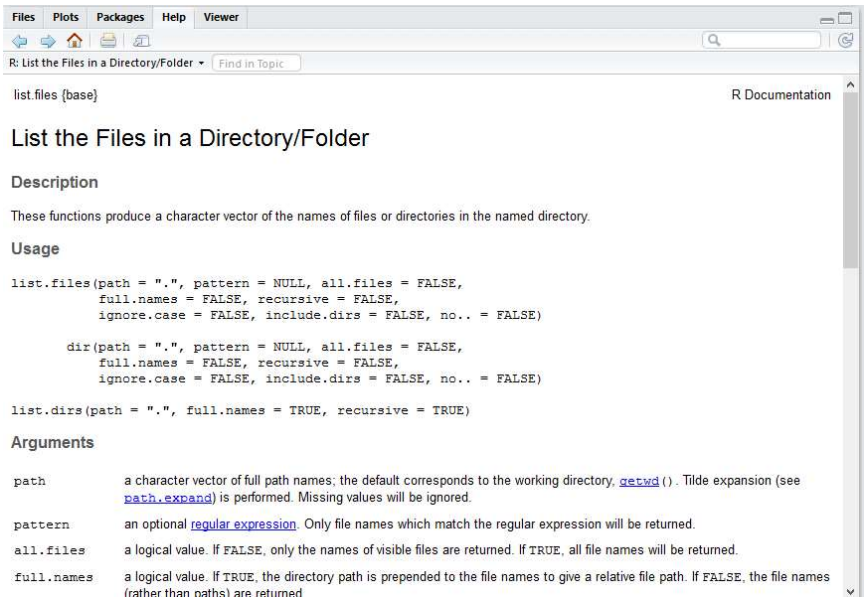
   When you run the script, this line will change the place R looks for data that will be processed by setting the working directory to the variable, path, which is the string that you assigned in the previous step. Your code should look similar to the image below.



3. Now try running the lines you just wrote. Remember you already ran the code on lines 5 and 6 in the previous section. To run lines 8 and 9, highlight the lines and click run. You can also highlight the lines and hit Ctrl+R. You'll see the lines run in the console, but also notice that the path variable appears in the "Values" section in the upper right of the RStudio window. See below.

The values window is where you can see variable values that you have set.

4. Now explore the list.files method. RStudio comes with a convenient help section to explain methods to you. There are four main sections on the RStudio window. The one on the lower left has 5 tabs to choose from: Files, Plots, Packages, Help, and Viewer. We will explore more of these tabs later, for now click **Help.**
5. In the upper right corner of this section of RStudio is a search bar. Click in the search bar and type **list.files.** Hit **Enter**. What displays is the documentation about this method, how to use it, and the required arguments. When using RStudio, you will find yourself using this Help section often, as it provides useful information about how to use tools.



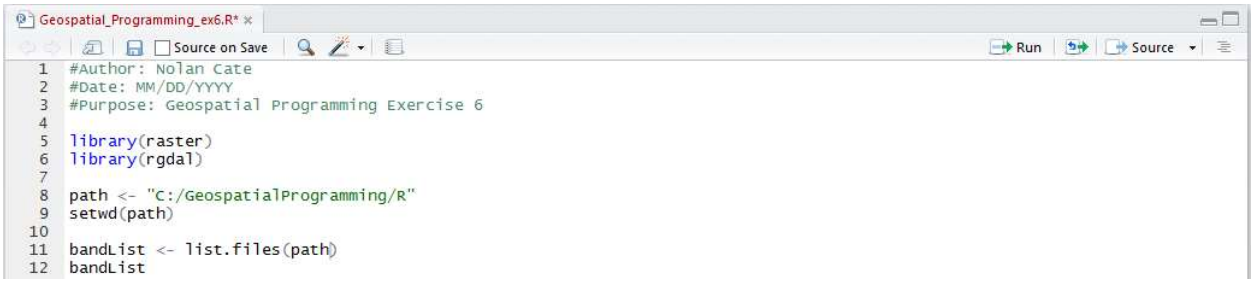
6. Back in the script you're writing, use the list files function to list the landsat bands in your course data folder. Create a variable called bandList and set it equal to the list.files() function with the path variable supplied as the location, by coding the line,

   `bandList <- list.files(path)`

7. On the next line, tell R to print out the files in the bandList variable, simply by typing

   `bandList`

   Your code should now look like this,



8. Now try running the two lines you just coded. Highlight the two lines beginning with bandList, and click **Run**. You will see a list with all of the files printed in your console. But there is a problem. This list contains more than just the Landsat bands in your course data folder, it contains a text file that isn't an image band. Filter the list to just include .tif files. You will also want the full path of the raster to be included in the name, not just the file name.

9. You need to specify a few other options in the list.files() function. Remember you can always review these options in the help section. Go back to the line, bandList <- list.files(path). In the parentheses after you supply the variable path, turn the option full.names to TRUE. The line should now read,

   `bandList <- list.files(path, full.names=TRUE)`

   You can try running the two lines again, to see the difference.

10. In the line, bandList <- list.files(path). In the parentheses after you specify full.names=TRUE, add a pattern for the list.files() function to look for. There are several different ways to do this. One common way is to use the glob2rx function. If you would like to read more about the glob2rx function, you can do so anytime in the help section in RStudio. For now, just

include a pattern specification to the list.files() function so that it looks for any files with the .tif extension. The line should now read,

```
bandList <- list.files(path, full.names=TRUE, pattern = glob2rx('*.TIF'))
```

11. Highlight the two lines beginning with bandList and click **Run.** Notice how the output in the console includes the full path names, and only includes the image files.

B. Create A Raster Stack

The raster package allows you to create R objects with imagery. The three main raster classes are raster(), stack(), and brick(). The raster() class is a single layer raster, while stack() and brick() can make multi-layer rasters. The difference is that brick() is meant to refer to a single multi-later file, while stack() creates a multi-layer raster object from many files. For more information see: https://cran.r-project.org/web/packages/raster/raster.pdf.

Like many things in scripting, there is not one way to do anything. In this section you will learn to use methods in the raster package, but there are other ways of exploring raster data in R.

1. You're going to load a true color image of your data using bands 4, 3, and 2.

Note: *You're using these bands because they represent the visible light portion of the spectrum. If you'd like to learn more about Landsat bands, you can do so here:*

https://landsat.usgs.gov/what-are-band-designations-landsat-satellites

Create a stack() object called rgbRaster and fill it with the list variable you made earlier in the exercise. Because we're only loading the red, green, and blue bands, you don't need to include all the bands in bandList to the stack. Type the line,

```
rgbRaster <- stack(bandList[4:6])
```

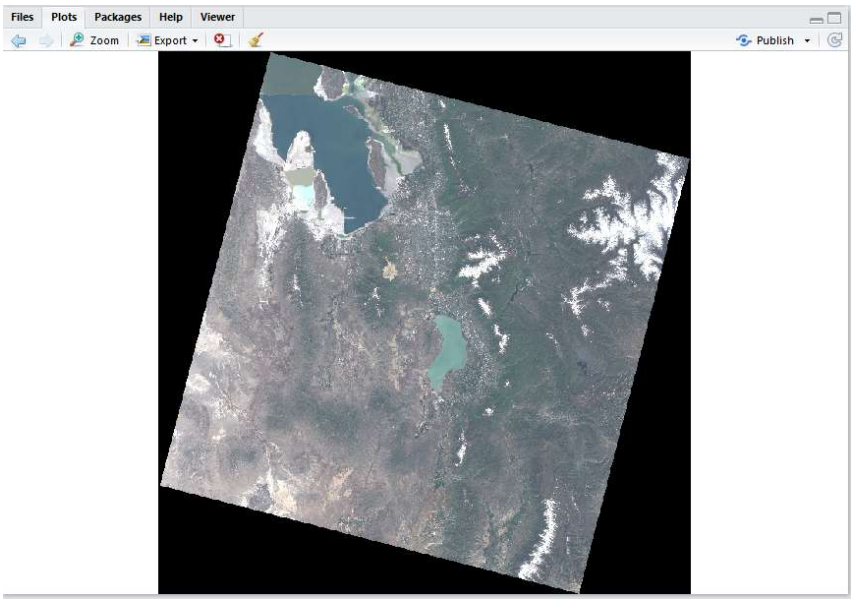
Notice that bands 4, 3, and 2 actually fell in positions 4, 5, and 6 in the bandList variable.

2. Displaying a true color image is easy using the plotRGB method in the raster package. If you want to read more about the plotRGB method, remember you can always read the documentation in the help section of RStudio. For now type the line,

```
plotRGB(rgbRaster, r=3, g=2, b=1, scale=30, stretch='lin')
```

This tells R to plot an RGB image from the rgbRaster variable, where red (r) is the third band in the stack, green (g) is the second, and blue (b) is the first. It also tells R to display the image at a 30 meter scale and apply a linear stretch for viewing.

3. Highlight the rgbRaster and plotRGB lines and click **Run**. This will take a few moments to run. Observe the output. Notice you'll need to be on the "Plots" tab in RStudio.



C. Image Calculations

You're going to calculate a Normalized Difference Vegetation Index (NDVI) image.

$$NDVI = \frac{Nir - red}{Nir + red}$$

To read more about NDVI, visit:
http://earthobservatory.nasa.gov/Features/MeasuringVegetation/measuring_vegetation_2.php.
This will give you the opportunity to practice using operators and objects in R.

1. The NDVI equation uses Red and Near Infrared (NIR) bands. Begin by creating raster objects for these bands. The NIR band is position 7 in your bandList variable, red is in position 6. Add the following two lines of code to your script.

```
nir <- raster(bandList[7])
red <- raster(bandList[6])
```

2. To ensure you've selected the correct bands, print the bands in the console. First you have to run the lines of code. Highlight the code you just wrote in step 1 and Click **Run.**
3. In the console, type **nir** and hit **Enter**. You should see the following printed out:

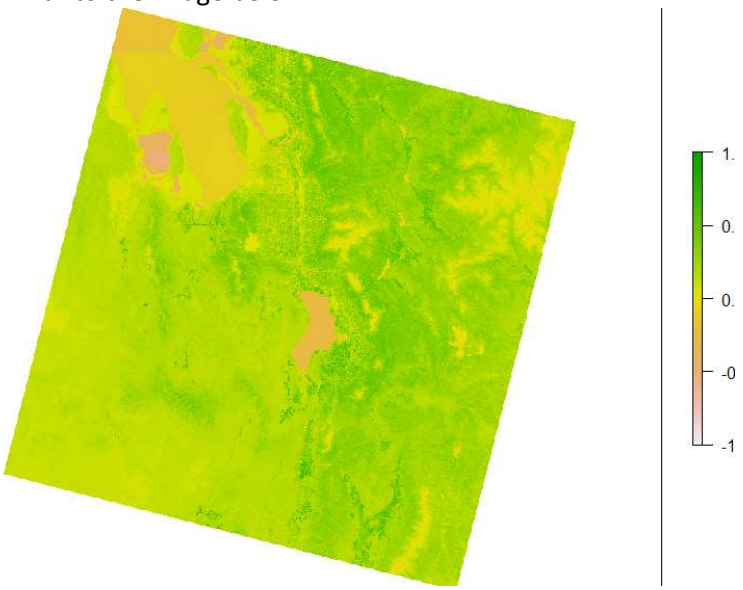
```
class       : RasterLayer
dimensions  : 7941, 7801, 61947741  (nrow, ncol, ncell)
resolution  : 30, 30  (x, y)
extent      : 302085, 536115, 4345485, 4583715  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=utm +zone=12 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0
data source : C:\GeospatialProgramming\R\LC80380322016153LGN00_B5.TIF
names       : LC80380322016153LGN00_B5
values      : 0, 65535  (min, max)
```

   Notice the property "names" shows that this is the band 5 layer. Repeat step 3 with the red band and ensure the band is band 4.
4. Now that you're sure you've created raster layers with the NIR and red bands, create an NDVI layer. Use the NDVI equation shown above to use. You can do this by using some simple operators. In your script, below the lines you wrote in step 1, write the line,

```
ndvi <- (nir-red)/(nir+red)
```

5. This will create a new raster layer with ndvi values.
6. Highlight the ndvi line in your script and click **Run**. This will create the ndvi variable.
7. Now that you've created an NDVI raster by applying some calculations to your raster, display the image. Use a plot() function. This is similar to what you learned in the previous section when you displayed an RGB image. In your script type

   `plot(ndvi, main='NDVI')`
8. Run the plot() function. It will take a few moments to run, then your output should look similar to the image below.



9. At the end of this section, your script should look similar to the image below:

Geospatial_Programming_ex6.R*

Source on Save

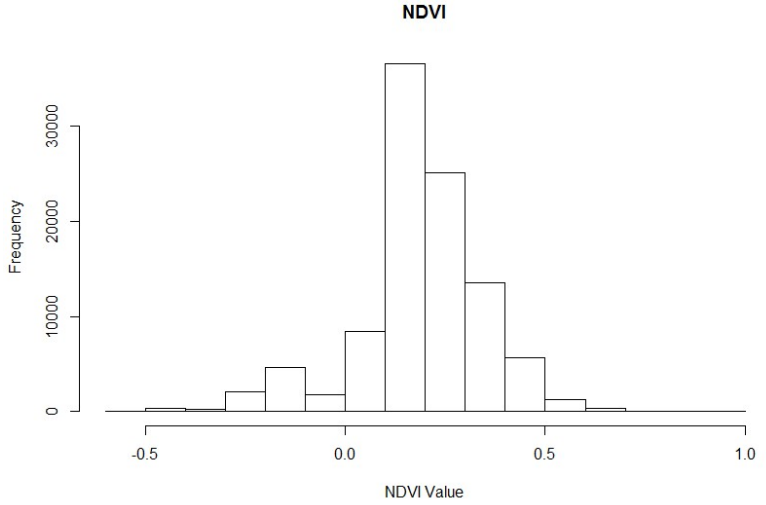
```
1  #Author: Nolan Cate
2  #Date: MM/DD/YYYY
3  #Purpose: Geospatial Programming Exercise 6
4
5  library(raster)
6  library(rgdal)
7
8  path <- "C:/GeospatialProgramming/R"
9  setwd(path)
10
11 bandList <- list.files(path, full.names=TRUE, pattern = glob2rx('*.TIF'))
12 bandList
13
14 rgbRaster <- stack(bandList[4:6])
15 plotRGB(rgbRaster, r=3, g=2, b=1, scale=30, stretch = "Lin")
16
17 nir <- raster(bandList[7])
18 red <- raster(bandList[6])
19 ndvi <- (nir-red)/(nir+red)
20 plot(ndvi, main='NDVI')
```

Part 3: Extracting Useful Information

So far you have explored raster data in R and you've done some useful calculations on your raster. But R is especially well known for its statistical capabilities. In this section you're going to learn how to visualize and extract some useful information from your rasters.

A. Generating A Frequency Histogram

1. Begin with a simple visualization by generating a histogram. In the console type
   ```
   help("hist,Raster-method")
   ```
2. Hit **Enter.** Read through the help section to learn more about the hist() method
3. Now back in your script, enter a line that will create a new histogram, with the title "NDVI" and an X axis label of "NDVI value". To do so, type
   ```
   hist(ndvi, main='NDVI', xlab='NDVI Value')
   ```
4. Run the hist() method you just wrote. It should produce a similar output to the image below.



There are other arguments that you can pass to this method to pull additional information from your data. You can experiment with selecting color options, or creating a density histogram

B. Adding Shapefiles and Tables

You're going to begin by loading points that you downloaded with course data. In your R folder that you downloaded, there is a shapefile called "Points". To load that shapefile in R, you're going to use a method in the rgdal library called readOGR.

Note: *the instructions below will walk you through how to use the readOGR method to load the shapefile. But before you go through them, take a moment to think about what you've learned about using R and RStudio today. Do you know where you could find help on how to use this method? Do you know where to look up what arguments you have to pass to this method?*

1. Create a variable called points and set it equal to the output of the readOGR method by typing, **points <- readOGR()**

2. Now in the parentheses you need to pass arguments to the readOGR method. The two arguments you need to use today is the dsn (data source name) and layer. In this case the dsn in the path to the location of the shapefile and the layer will be the shapefile called "points.shp". Add the arguments dsn='C:\\GeospatialScripting\\points', layer='points' to the argument so the full line now reads:

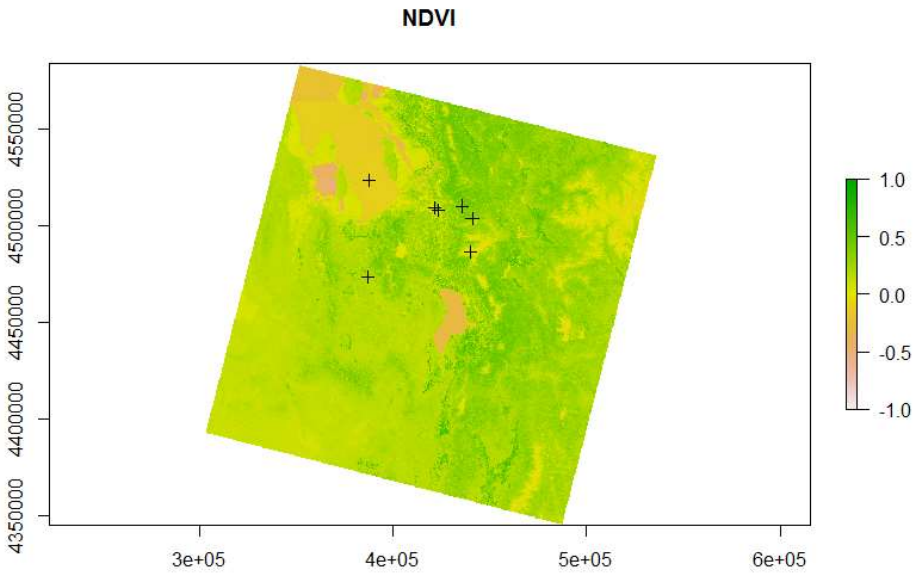
```
points <- readOGR(dsn='C:\\GeospatialScripting\\points', layer='points')
```

Remember that in R the file path needs to have the double backslashes (\\) to recognize the file path.

3. Run the line. Notice the points variable will get added to your values window in RStudio.
4. Now plot the points so that you can see them over the NDVI image that you generated previously. Run the **plot(ndvi, main='NDVI')** line again. You can do this by placing your cursor on the line you wrote earlier and hitting Ctrl+R, there is no need to rewrite the line.
5. The NDVI image will appear in your plots tab in RStudio. Now you can add a plot line to your script to add the points over the top of that image. You can use the same plot command by typing plot(points), but this time use an extra argument called add. Add is a logical argument so its value can be true or false. In this case we want the value to be true, because we just want to add the points to the existing plot. The line should read

   ```
   plot(points, add=TRUE)
   ```

6. Run the line. You should see the NDVI image remain in the plot window, with 6 points added to the window, similar to the image below.



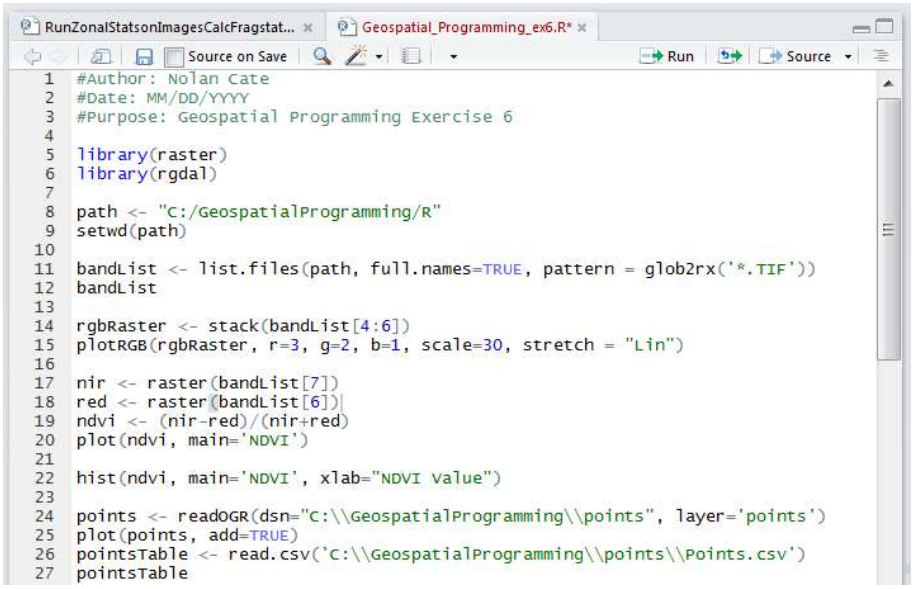
7. One last variable you should bring in to RStudio is a csv that lists these points. You can examine the csv in your GeospatialScripting folder, in the R subfolder, called Points.csv. To load this csv file in RStudio you can use the command read.csv(). Create a new variable called pointsTable and set it equal to read.csv(), where the argument you pass to read.csv is the path to the Points csv file:

```
pointsTable<-read.csv('C:\\GeospatialScripting\\R\\points\\Points.csv')
```

8. Run the line. Notice that the pointsTable variable will be added to your values window.
9. Print the pointsTable variable by typing **pointsTable** either into your console, or in your script and running the line. It should output information similar to the data seen below.

```
> pointsTable <- read.csv('C:\\GeospatialProgramming\\points\\Points.csv')
> pointsTable
  FID  POINT_X POINT_Y
1   0 387418.5 4523662
2   1 439962.5 4486519
3   2 435644.5 4510141
4   3 423537.2 4508193
5   4 421251.2 4509040
6   5 440893.9 4503791
7   6 386707.1 4473565
> |
```

The POINT_X and POINT_Y columns are the location of the points.
At this point your script will look similar to the image below.



C. Adding NDVI Values to the CSV

There are a number or situations when extracting values at a given location from a raster may be useful. For example, if you were creating a biomass model from lidar data you might need to measure biomass at a point on the ground, then associate that measurement with the value of a raster. The general workflow for that process is outlined below.

1. Your first step it to extract the NDVI values at the 6 points that you've loaded up. Create a new variable called ndviValues and use the extract method to extract a raster value. There are two arguments you need to pass to the extract method. The first is the raster object, the second is the point locations. With this information, could you write the necessary line of code without needing the instructions below?

   Type the line,
   `ndviValues <- extract(ndvi, points)`
2. Run the line. Notice that the ndviValues variable appears in your variables window.
3. Now print the ndviValues variable. Either type **ndviValues** in the console, or type the line in your script and run the line. This will print a list of the ndviValues at the 6 points.
4. You're going to place the ndviValues variable in a new data frame with the points csv that's in the pointsTable variable.

Note: *a data frame is an object in R that is used to store data tables. Because we're going to be working with a data table holding location values and ndvi values, you're going to create a new data frame object. You can read more about data frames in R here:*

http://www.r-tutor.com/r-introduction/data-frame

To do this, create a new variable called ndviFrame and create a new data frame by typing the line:

```
ndviFrame <- data.frame(pointsTable, ndviValues)
```

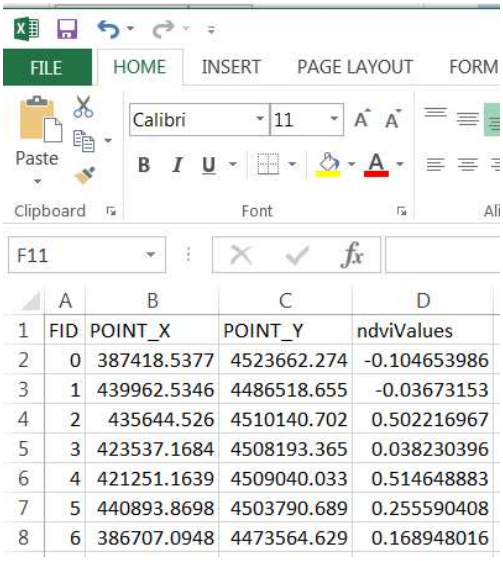
5. Run the line of code. Notice that the ndviFrame variable appears in your data window.
6. Print the ndviFrame variable by typing ndviFrame into the console or typing the line in your script and running the line. This will show the new data frame you've created. See the image below.

```
> ndviFrame <- data.frame(pointsTable, ndviValues)
> ndviFrame
  FID  POINT_X POINT_Y  ndviValues
1   0 387418.5 4523662 -0.10465399
2   1 439962.5 4486519 -0.03673153
3   2 435644.5 4510141  0.50221697
4   3 423537.2 4508193  0.03823040
5   4 421251.2 4509040  0.51464888
6   5 440893.9 4503791  0.25559041
7   6 386707.1 4473565  0.16894802
>
```

7. Notice that this data frame looks like the pointsTable variable, but with the ndviValues added. This is what we want to write to our csv, which is our last step. Writing to a csv is simple. Write the line in your script:

```
write.csv(ndviFrame,file="C:\\GeospatialScripting\\points\\
Points_fromR.csv", row.names = FALSE)
```

8. Be aware this is all one line in RStudio, not two. This tells R to write a csv, called Points_fromR.csv in the folder C:\GeosptialScripting\Points. And because you added the NDVI values to the data frame, those will be included in your new csv. Run the line.
9. In windows explorer, navigate to that folder and notice that you will have a new csv file called Points_fromR. Double click the file to open it.



	A	B	C	D
1	FID	POINT_X	POINT_Y	ndviValues
2	0	387418.5377	4523662.274	-0.104653986
3	1	439962.5346	4486518.655	-0.03673153
4	2	435644.526	4510140.702	0.502216967
5	3	423537.1684	4508193.365	0.038230396
6	4	421251.1639	4509040.033	0.514648883
7	5	440893.8698	4503790.689	0.255590408
8	6	386707.0948	4473564.629	0.168948016

10. This is a good way to pull values from a raster at a given location.

Part 4: Notes and Other Resources

- GTAC offers additional trainings in R.
  - The course *Geospatial Scripting in R* is under development. Look for it later in 2017!
  - The tutorial for the course *Using R to Explore Inventory Modeling with Lidar* is available here: http://fsweb.geotraining.fs.fed.us/www/index.php?lessons_ID=2964
- There are many other places to find documentation on R. Official documents can be found here: https://www.r-project.org/other-docs.html

Congratulations! You have learned some basic and very important skills about geospatial scripting using R. This should have given you the chance to practice some of the skills you learned in earlier exercises and given you some resources to continue learning. Remember that when scripting you are learning a new language, and it will take time. Review this exercise and visit the additional resources provided to keep practicing your new skills.